LA-UR-01-1804
Approved for public release; distribution is unlimited.

Title:

A PARALLEL MULTIGRID METHOD FOR INVERSION OF THE DIFFUSION OPERATOR IN NEUTRONICS APPLICATIONS

Author(s): Raymond E. Alcouffe

Submitted to:

http://lib-www.lanl.gov/la-pubs/00818579.pdf

A PARALLEL MULTIGRID METHOD FOR INVERSION OF THE DIFFUSION OPERATOR IN NEUTRONICS APPLICATIONS

Raymond E. Alcouffe
Transport Methods Group
Los Alamos National Laboratory
Los Alamos, NM 87545
(rea@lanl.gov)

Abstract

The multigrid method has been shown to be the most effective general method for solving the multi-dimensional diffusion equation encountered in neutronics. This being the method of choice, we develop a strategy for implementing the multigrid method on computers of massively parallel architecture. This leads us to strategies for parallelizing the relaxation, contraction (interpolation), and prolongation operators involved in the method. We then compare the efficiency of our parallel multigrid with other parallel methods for solving the diffusion equation on selected problems encountered in reactor physics.

1. INTRODUCTION

In this work we study the parallelization of the multigrid method for solving multidimensional diffusion problems. The reason for doing this is twofold: (1) the main computational platform for the foreseeable future will have a massively parallel architecture, and (2) the most effective method for inverting the multidimensional diffusion operator encountered in neutronics is the multigrid method (Alcouffe, et al., 1981) and (Alcouffe, 1983). However the task at hand, parallelizing the inversion of the diffusion operator is a formidable one since we also desire to do this in an efficient and scalable manner. A scalable algorithm is one which as the number of processors increases, the computational time decreases proportionately. Because of the use of multiple grids and the need to communicate between them, the multigrid method inherently is difficult to make scalable. Our context in attacking the communications problem is message passing, and we use the methods incorporated into MPI on our machine.

In the following we outline the steps required for the multigrid method for inverting the diffusion operator and we investigate how to make these steps parallel. We then select some problems to measure and demonstrate our degree of success. The implementation has been carried out in the PARTISN code (Alcouffe, et al., 2000).

1.1 Discretization of the Diffusion Equation

In this paper we focus on neutronics where the diffusion problem is generally a multigroup one which necessitates the solution of a diffusion equation for each energy group. In addition, for fission problems we need to do outer iterations; thus, we invert the diffusion operator many times in the course of the solution of neutronics problems. Efficiency in the inversion of the diffusion operator is therefore critical and we attempt to minimize the computational effort by choosing as simple a differencing of the diffusion operator as possible. The simplest form of the finite differenced diffusion operator has a five point stencil in two dimensions and a seven point stencil in three dimensions. Also in order to be compatible with the DSA method, we take the unknowns to be on the mesh vertices. To begin we write the diffusion equation in the following form:

$$-\nabla \bullet D(\vec{r})\nabla\phi(\vec{r}) + \Sigma_R(\vec{r})\phi(\vec{r}) = Q(\vec{r}) \tag{1}$$

where

D is the diffusion coefficient,

Σ_R is the removal cross section,

Q is the source, and

ϕ is the flux to be solved for.

The vertex centered finite difference form of this equation is written as follows:

$$\begin{gathered} -H_{i+1}\left(\phi_{i+\frac{3}{2}} - \phi_{i+\frac{1}{2}}\right) + H_i\left(\phi_{i+\frac{1}{2}} - \phi_{i-\frac{1}{2}}\right) - V_{j+1}\left(\phi_{j+\frac{3}{2}} - \phi_{j+\frac{1}{2}}\right) + V_j\left(\phi_{j+\frac{1}{2}} - \phi_{j-\frac{1}{2}}\right) - \\ -F_{k+1}\left(\phi_{k+\frac{3}{2}} - \phi_{k+\frac{1}{2}}\right) + F_k\left(\phi_{k+\frac{1}{2}} - \phi_{k-\frac{1}{2}}\right) + (\Sigma_R W\phi)_{i+\frac{1}{2},\, j+\frac{1}{2},\, k+\frac{1}{2}} = \\ (QW)_{i+\frac{1}{2},\, j+\frac{1}{2},\, k+\frac{1}{2}} \end{gathered} \tag{2}$$

where

H, V, and F are the horizontal, vertical, and front-back leakage coefficients derived from D and the mesh cell sizes, and

W is the cell volume.

1.2 The Multigrid Method

Briefly, the multigrid method for this work makes use of a series of grids which are a subset of what is called the fine mesh grid used to discretize the diffusion equation. The coarser grids are such that there are two finer-grid mesh intervals within each coarse mesh interval in each coordinate direction. An allowance is made for the last coarse mesh interval to be the same size as the fine mesh interval if the number of finer mesh intervals is not evenly divisible by 2. An example of a series of grids is shown in Fig. 1 below. The coarser mesh points are designated by X in the figure and the coarsest mesh points are those designated by ✯.

To layout the multigrid method, we write the discretized diffusion equation on the finest grid in operator notation as:

$$L^n \phi^n = Q^n \tag{3}$$

Assuming that we don't invert the solution on the finest grid, n, exactly, we form the residual on that grid as;

$$R^n = Q^n - L^n \phi^n$$

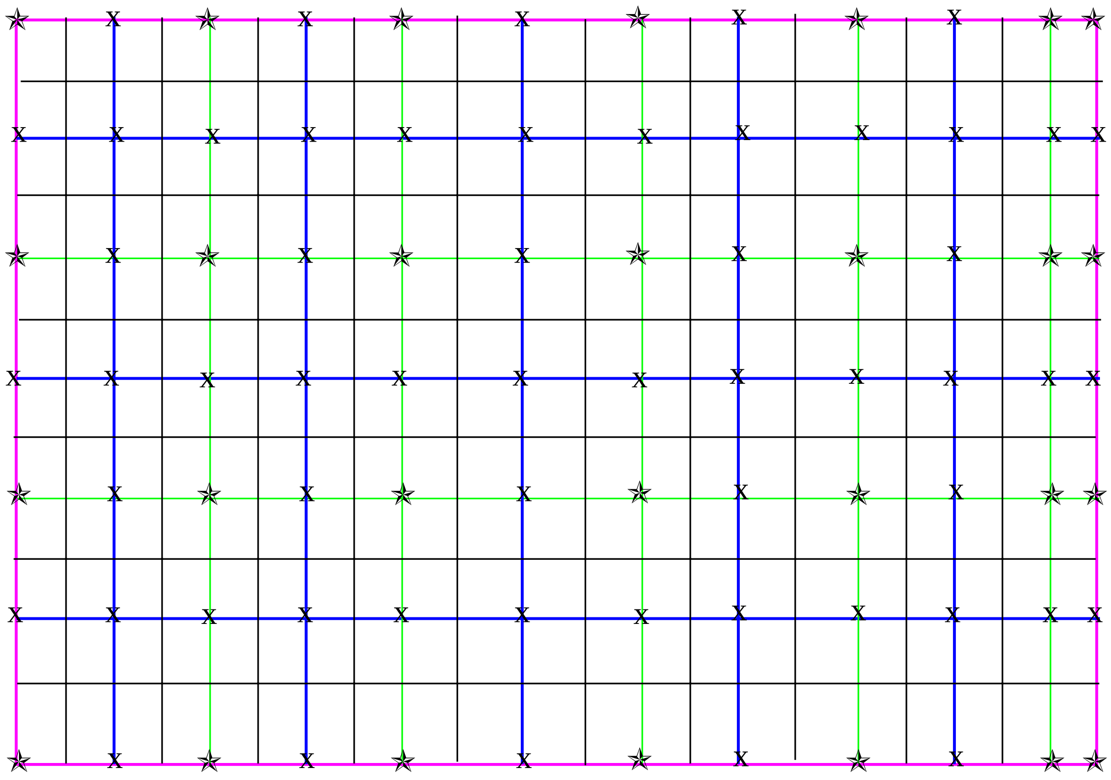

Fig. 1 An example 3 level grid for the diffusion equation.

We then form the correction equation on the coarser grid, n-1, as:

$$L^{n-1} \varphi^{n-1} = R^{n-1} \tag{4}$$

where

$$R^{n-1} = I_n^{n-1} R^n$$

$$L^{n-1} = I_n^{n-1} L^n I_{n-1}^n$$

I_{n-1}^n is the interpolation operator, and

I_n^{n-1} is the prolongation operator which is derived from the interpolation.

The interpolation operator is defined to explicitly account for large discontinuities in the diffusion coefficient, D, and is a linear interpolation based upon the continuity of the diffusion currents, $-D\nabla\phi$. The prolongation operator is adjoint to the interpolation operator; both are given in (Alcouffe, et al., 1981). Since the solution to Eq. (4) is the correction on the coarser grid to the solution on the fine grid, the corrected fine grid solution is thus:

$$\tilde{\phi}^n = \phi^n + I_{n-1}^n \varphi^{n-1} \tag{5}$$

In order to solve Eq. (4) we then use a still coarser grid, n-2, and follow the same procedure used to obtain Eq. (4). This results in a series of grids from the finest to the coarsest to develop corrected solutions on the next finer grid. The solution for the correction on the coarsest grid is obtained by direct inversion.

The multigrid method as used to invert the diffusion operator can thus be described by the following procedure:

- **a.** Do a relaxation sweep over the fine grid; we have found that the most efficient relaxer is line relaxation in each of the 3 coordinate directions
- **b.** Form the residual and transfer it to the next coarser grid using the prolongation operator.
- **c.** Do a relaxation sweep over this next coarser grid using the same relaxation method as for the fine grid.
- **d.** Repeat steps b and c until the coarsest grid is reached. On the coarsest grid do a direct inversion of the diffusion operator to obtain a solution.
- **e.** Interpolate the solution from the coarsest grid onto the next finer grid and add it to the correction obtained from the relaxation step b.
- **f.** Do a relaxation over this grid to smooth the correction.
- **g.** Repeat steps e and f until the finest grid is obtained.

The procedure outlined above is called a V-cycle for the multigrid method. Once the finest grid has been attained in this procedure, the error (L2 or L-infinity norm) is computed and a test is made to determine if the error has been sufficiently reduced to exit the procedure. If not, another V-cycle is performed until convergence is attained.

1.3 PARTISN Parallelization

The parallelization technique in PARTISN is based upon spatial domain decomposition. The X direction mesh quantities lie entirely within a processor, but the Y and Z directions are divided into subdomains which are distributed over a two-dimensional processor space. The layout for a 16 processor decomposition is as depicted in Fig. 2 below. The numbers in the cells indicate the processor number that contains its portion of the spatial mesh in the Y-Z plane. This arrangement has ramifications in the solution of the diffusion equation by the multigrid method. In the 1 processor case it was noted that line relaxation in each of the coordinate directions was the preferred relaxer. In the parallel case with the current spatial decomposition, a line relaxation in the X direction entails no data transfers in the direction of the line sweep; however, in the Y and Z directions, data communications are necessary and will have an impact on the performance and scalability of the multigrid method.

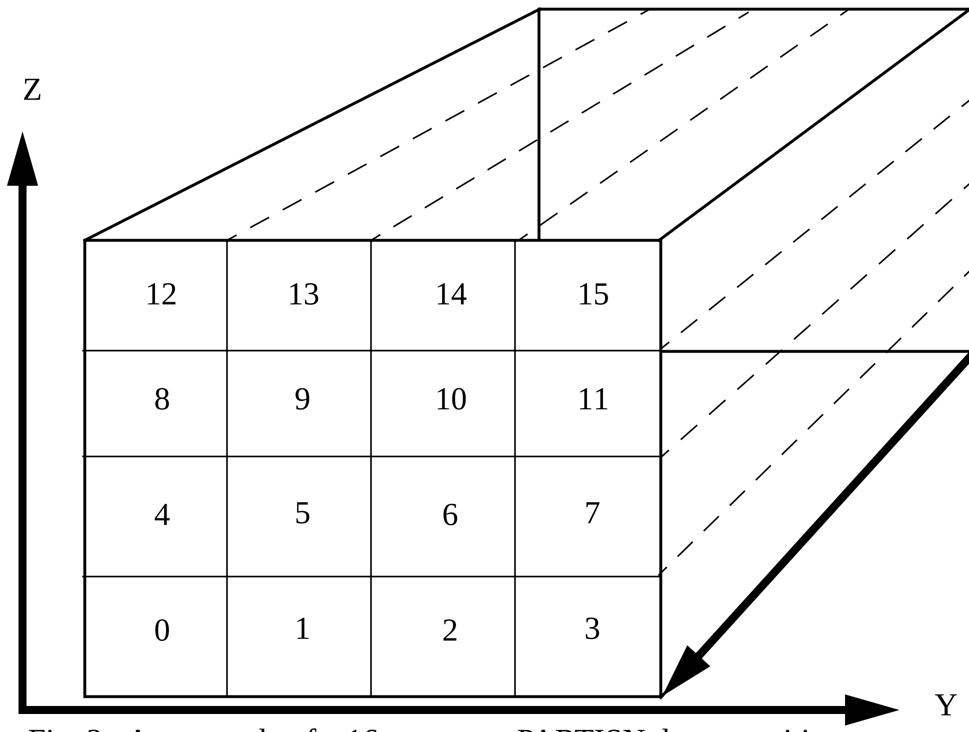

Fig. 2 An example of a 16 processor PARTISN decomposition.

2. PARALLELIZATION OF THE MULTIGRID METHOD COMPONENTS

In order to make the multigrid method parallel, we must consider the parallelization of each of the steps outlined in section 1.2 in the light of our spatial mesh decomposition. That is, we must parallelize the generation of the coarse mesh coefficients for each grid, the application of the prolongation operator to lay down the residuals, the interpolation operator, and finally the relaxation method. In the parallelization of the relaxation on each of the meshes, we will consider 2 methods: (1) line relaxation in each of the coordinate directions, and (2) line relaxation in the X direction only with a red-black updating of the flux in the Y and Z directions. We will discuss some refinements of option 2 in the section on relaxation with respect to our approach of load balancing.

2.1 A Load Balancing Technique for Multigrid

We note that with the spatial decomposition described in Section 1.3 and the fact that the number of mesh intervals on each grid decreases by a factor of 8 from finer to coarser grid, we will reach a condition where the number of mesh intervals for a grid on each processor will be too few for efficient computation. Thus in the interest of load balancing in the sense that we desire to keep the ratio of communication to computation effort the same on the coarse meshes as on the finest mesh, we subdivide the processor space such that each successive coarse grid will occupy a smaller subset of the processors that the finer grids. Referring to Fig. 2, we condense the processor space by a factor of 2 in the Y and Z directions as we coarsen the grid. In Fig. 3 we give an example of such a processor partitioning.

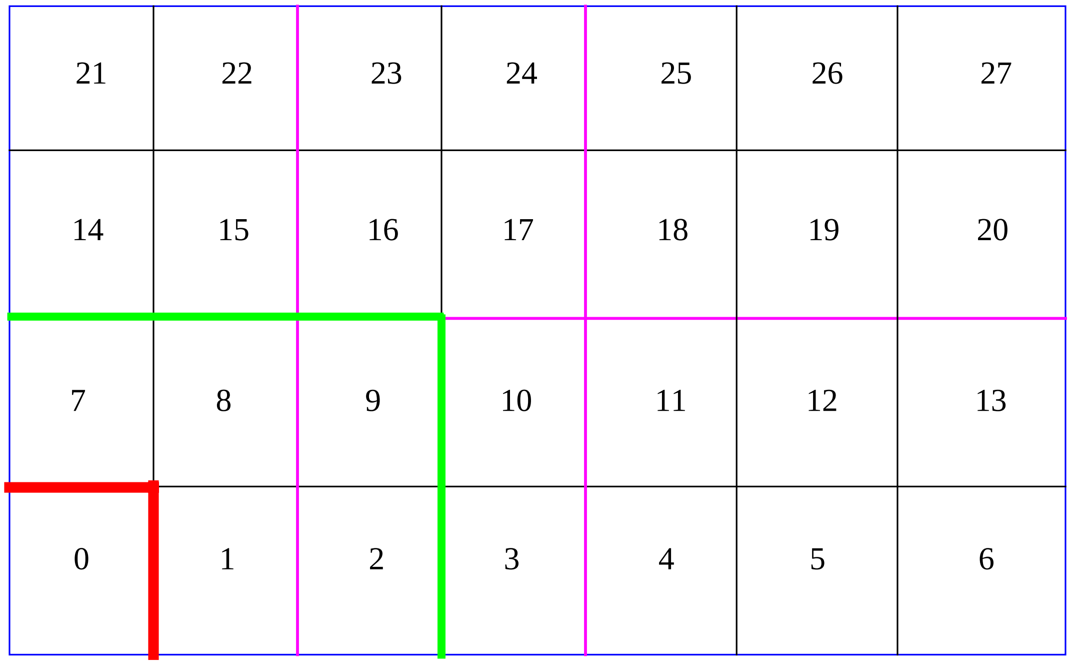

Fig. 3 Processor arrangement for finest to coarser grids.

In this example, the finest grid is partitioned onto 28 processors; the next coarser grid is partitioned onto 9 processors; and the remaining grids all reside on PE 0. Thus if we are going from the finest to the coarsest mesh, the information of the finest mesh must be transferred from its processors to the 9 processors of coarser grid. Thus in the example above, the information on PE’s 0, 1, 7, and 8 goes to PE 0; the information on 2, 3, 9, and 10 goes to PE 1; that on PE’s 4, 5, 6, 11, 12, and 13 goes to PE 3; that on PE’s 14, 15, 21, and 22 goes to PE 7; that on PE’s 16, 17, 23, 24 goes to PE 8; and that on PE’s 18, 19, 20, 25, 26, and 27 goes to PE 9. Then going from that coarse grid to the next coarsest, all the information on PE’s 0, 1, 2, 7, 8, 9 goes to PE 0. In this way the amount of work done on each grid remains approximately the same on a per PE basis except for the factor of 2 in the X direction. This process is reversed in going from the coarser grids to the finer which is basically the interpolation operation. Of course while that work is being done on the receiving PE’s, the others remain idle. There is also the overhead of moving the data from the sending to the receiving PE’s. We will show the impact of this on the efficiency of the multigrid algorithm in our examples below.

2.2 Generating the Coarse Mesh Diffusion Operator in Parallel

To indicate how this is done, we derive the coarse mesh leakage operator from the fine mesh operator, H_i. We first form an intermediate quantity on the fine grid in the Y and Z directions, Hx_{ic}:

$$Hx_{ic} = \frac{H_i H_{i+1}}{H_{i+1} + H_i} \tag{6}$$

This quantity is formed on all the fine mesh processors. The coarse mesh coefficient is then formed on all the fine mesh processors by doing the following operation:

$$H_{ic,\,jc,\,kc} = Hx_{j,k} + \frac{1}{2}(Hx_{j-1,k} + Hx_{j+1,k}) + \frac{1}{2}\left[Hx_{j,k-1} + \frac{1}{2}(Hx_{j-1,k-1} + + Hx_{j+1,k-1})\right] + \frac{1}{2}\left[Hx_{j,k+1} + \frac{1}{2}(Hx_{j-1,k+1} + Hx_{j+1,k+1})\right] \tag{7}$$

In Eq. (7) the j and k indices are the fine mesh while jc and kc are the coarse mesh indicies. Of course, Eq. (7) is evaluated at each coarse mesh ic. We note that at processor boundaries, we need to move the leakage data from adjacent processors in order to account for the j-1, j+1, k-1, and k+1 fine mesh quantities appearing the Eq. (7). This is done by an MPI send-receive operation over the mesh on the processor faces. Once the coarse mesh operator has been evaluated, it must be moved from the fine mesh processors to the coarse mesh processors. The other leakage coefficients are treated in a way that is similar to that done in Eq. (7). The removal term is treated like that given to the prolongation operator described below.

2.3 The Prolongation Operator in Parallel.

The prolongation operator, I_n^{n-1}, is one that lays down the fine mesh quantity onto the coarse mesh much like that shown in Eq. (7). If the fine mesh quantity is $R_{i,\,j,\,k}$, then the prolongation is:

$$R_{ic,\,j,\,k} = R_{i,\,j,\,k} + \frac{1}{2}(R_{i+1,\,j,\,k} + R_{i-1,\,j,\,k}) \tag{8}$$

$$R_{ic,\,jc,\,kc} = R_{ic,\,j,\,k} + \frac{1}{2}(R_{ic,\,j-1,\,k} + R_{ic,\,j+1,\,k}) + \frac{1}{2}\left[R_{ic,\,j,\,k-1} + \frac{1}{2}(R_{ic,\,j-1,\,k-1} + + R_{ic,\,j+1,\,k-1}) + R_{ic,\,j,\,k+1} + \frac{1}{2}(R_{ic,\,j-1,\,k+1} + R_{ic,\,j+1,\,k+1})\right] \tag{9}$$

As a parallel implementation, Eq. (8) is done on each processor to accumulate the intermediate values. Then Eq. (9) is evaluated on each processor with data being moved on the processor boundaries in the Y and Z directions as appropriate. When R has been computed on the fine mesh processors, it is then moved to the coarse mesh processors in the same way as the coarse mesh coefficients are.

2.4 The Interpolation Operator in Parallel.

The interpolation operator takes the coarse mesh solution and puts it on the finer mesh. In order to do this the discretized diffusion operator is utilized to better account for discontinuities in the diffusion coefficient. In this vein, interpolation is done first along lines of the mesh, then on faces, and finally the center point of each coarse mesh. This is illustrated in Fig. 4:

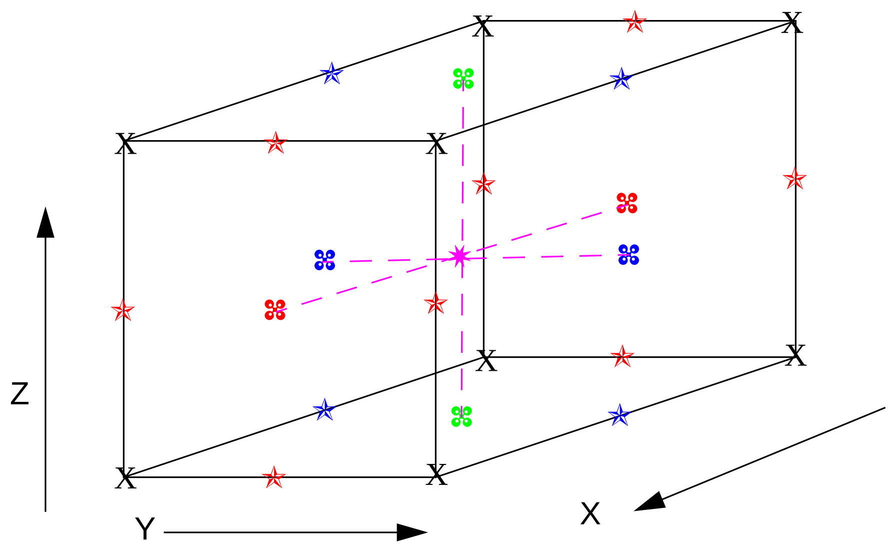

Fig. 4 The interpolation of the coarse mesh solution to the finer mesh.

In Fig. 4, the X’s are the coarse grid points and the colored symbols denote the sequence of interpolations. To do the interpolation in parallel, we first go through the Y-Z planes defined by the coarse grid on each processor and interpolate in lines along Y and then in lines along Z. These are the red ★ symbols in the diagram. We then do the interpolation on the Y-Z faces denoted by the ⌘ symbols. Next we do the interpolations along lines in X for the coarse X-Y faces as denoted by the symbol ★. This is followed by interpolations on the X-Y faces denoted by ⌘. This is followed by interpolations on the X-Z faces (⌘). Once these are done then the body centered point (✺) can be obtained by interpolation from the 6 face centered points. To do this in parallel requires a consideration of the processor boundaries which involves moving coarse mesh information on the processor boundaries. The fine mesh solution is then moved from the coarse mesh processors to the finer mesh processors. To illustrate the interpolation operator for lines along the Y direction, we have:

$$\varphi_{jf} = \frac{V_{jf}\varphi_{jc} + V_{jf+1}\varphi_{jc+1}}{V_{jf} + V_{jf+1}}$$

where jc refers to the coarse mesh points and jf the fine mesh points. On the Y-Z face

$$\varphi_{jf,kf} = \frac{V_{jf}\varphi_{jf-1,kf} + V_{jf+1}\varphi_{jf+1,kf} + F_{kf}\varphi_{jf,kf-1} + F_{kf+1}\varphi_{jf,kf+1}}{V_{jf} + V_{jf+1} + F_{kf} + F_{kf+1}}$$

The center point, (✲), uses the six face points and the corresponding six leakage coefficients in each direction.

2.5 A Parallel Relaxation Method.

In the multigrid method, the purpose of the relaxation step is to damp the high frequency error as measured on the grid level being solved. Experience has shown that this is most effectively done by doing line relaxation along each of the coordinate directions. We illustrate this procedure by examining the method used to do line inversion in the X direction. We form an iterative procedure where we move the off line components of the diffusion operator of Eq. (2) to the right hand side (RHS) as:

$$-H_{i+1}\left(\phi^{l+1}_{i+\frac{3}{2}} - \phi^{l+1}_{i+\frac{1}{2}}\right) + H_i\left(\phi^{l+1}_{i+\frac{1}{2}} - \phi^{l+1}_{i-\frac{1}{2}}\right) + ([\Sigma_R W] + V_j + V_{j+1} + F_k + F_{k+1})\phi^{l+1} =$$

$$= (QW) + V_{j+1}\phi^{l}_{j+\frac{3}{2}} + V_j\phi^{l}_{j-\frac{1}{2}} + F_{k+1}\phi^{l}_{k+\frac{3}{2}} + F_k\phi^{l}_{k-\frac{1}{2}}$$

where l is the red-black iteration index. (10)

The steps in the solution of Eq. (10) is thus to form the right hand side which in the multiprocessor case involves MPI send-receives at the processor boundary for our spatial decomposition. The solution is then found by inverting the left hand side in a forward elimination - backwards substitution method that is the normal procedure for a tridiagonal solver. In the parallel case once the RHS has been formed, the solution is scalable with number of processors since all of the i mesh are on each processor. This is not true for lines in Y and Z and thus line relaxation in those directions is not very efficient. We show that in our examples. We designate this method as MG, i.e., full multigrid.

As an alternative, we do red-black points over the mesh as a relaxation method with line inversion in the X direction only. Thus as shown in Fig. 5 the red-black arrangement of the lines in the Y-Z plane allows the computation of the RHS using the current values of the solution which means that all the either red or black lines can be solved simultaneously. Thus we have a parallel algorithm for the line relaxation in the X direction. We find that for many of the problems that we have solved, it is sufficient to do this solve by sweeping over the Y-Z plane 2 times per grid to get good performance out of the multigrid algorithm. We designate this method as MG1L, i.e., multigrid with line sweeps in X only.

For completeness we also have two preconditioned conjugate gradient methods to

```
R B R B R B R B R B R B R B R B
B R B R B R B R B R B R B R B R
 R B R B R B R B R B R B R B R B
 B R B R B R B R B R B R B R B R
 R B R B R B R B R B R B R B R B
```

Fig. 5   Line arrangement on the Y-Z plane for relaxation.

solve the diffusion equation. The conjugate gradient method is selected because it is easily made parallel. The preconditioning is necessary for good performance of the method in the general case. The first preconditioner we use is that described above for the multigrid relaxation - we use line inversion in the X direction with a red-black sweep of the Y-Z plane. This method we term CG1L. The second preconditioner is to use the MG1L method above which we designate as CGMG.

3. EXAMPLE DIFFUSION PROBLEMS

We have chosen two problems to demonstrate some of the properties and the performance of the parallel multigrid method while comparing to the current method for solving the diffusion equation in PARTISN. The first problem is based upon our favorite shielding problem, what we call the iron-water shield problem. We use this to lay out some of the basic performance issues for a source driven, 3D diffusion problem. The second problem is based upon a small light water reactor on which we seek to compute the k eigenvalue of the system. This is a more challenging diffusion problem because of the fissions and it is highly heterogeneous (Baker, 1997). We use this to demonstrate that our observations on the simple shield problem carry over to a more complicated 3D criticality problem.

3.1 Diffusion Solution of the Iron-Water Shield Problem.

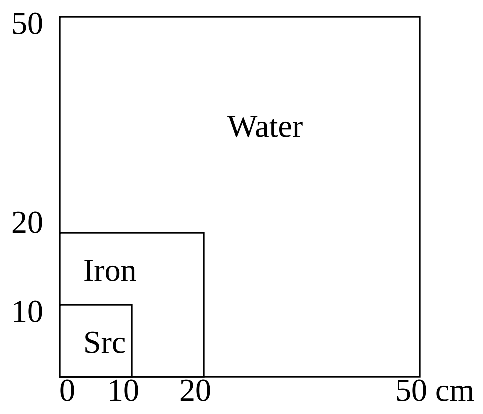

Fig. 6 Iron-water shield schematic.

A 2D schematic of this problem is given in Fig. 6 and the 3D version is a symmetric extension of this in the Z direction. The source region contains a uniform spatial source with a 3 group energy distribution given as 0.739, 0.261, 0.0. Given this source and the 3 group cross sections, we are able to solve the diffusion problem. We choose to solve this using the multigrid with the X-line sweep as the relaxer (MG1L). We compare this with the conjugate gradient method that is standard in PARTISN which has as a preconditioner the same X-line sweep with red-black updating in the Y-Z plane (CG1L). The one processor version of the spatial mesh used to solve the problem is a uniform 1 cm mesh throughout, thus 50x50x50 cells. The number of mesh intervals is changed on the multiprocessor problems so as to maintain the same number of mesh intervals on each processor. Since the extent of the problem geometry remains the same, this means that the mesh is refined in going from 1 to 126 processors; the latter thus has 16 million mesh intervals (252x252x252). In Tables 1 and 2 we present some selected results on the time spent in various portions of the solution.

TABLE 1. Iron-Water Shield Data from a Parallel Multigrid Solver

	Multigrid Component Times (secs)					Work	Grind
Procs	Total	T_diff	T_rlx	T_int	T_prl	Units	Time (ns)
1	9.3	7.44	5.06	1.11	0.51	74.7	994.0
2	13.5	11.28	7.84	1.67	0.71	76.9	700.5
4	17.2	14.87	10.43	2.04	1.10	79.0	441.2
8	17.1	14.70	9.15	2.87	1.27	81.0	211.0
16	19.9	17.24	10.88	3.17	1.44	81.0	122.6
32	23.1	20.26	12.12	4.34	1.87	89.6	64.0
64	22.7	19.70	11.77	4.10	1.76	85.3	33.3
126	33.8	29.25	16.12	6.69	3.21	85.3	25.1

TABLE 2. Iron-Water Shield Data from a Parallel Conjugate Gradient Solver

	CG Component Times (secs)					Work	Grind
Procs	Total	T_diff	T_rlx	T_int	T_prl	Units	Time (ns)
1	20.3	18.56	11.50	0.0	0.0	159	1019.3
2	34.0	32.03	19.56	0.0	0.0	191	711.7
4	41.1	38.84	21.76	0.0	0.0	243	342.9
8	49.5	47.60	25.86	0.0	0.0	298	166.2
16	61.8	59.80	33.47	0.0	0.0	367	84.2
32	80.6	78.60	42.69	0.0	0.0	460	43.6
64	154.7	151.97	58.81	0.0	0.0	562	34.4
126	284.7	273.81	93.33	0.0	0.0	696	25.8

Column labels are: Procs, the number of processors used, Total, the total solution time, T_diff, the time spent in inverting the diffusion operator, T_tlx, the time devoted to relaxing over the mesh, T_int, the interpolation time, and T_prl, the time for the prolongation operation. The remaining time, not shown, is the diffusion setup time which includes the line relaxation precalculations and the CG operations for the CG method. The grind time is the time expended in the solution per spatial mesh cell and a work unit is measured in terms of an X-line relaxation over the entire mesh of the problem.

Upon examining the tables, one thing that is immediately striking is that the multigrid method is very stable on the amount of work that is being done to develop the solution as the mesh size is decreased. The work unit referred to in the tables is measured as the computational work expended to do one line sweep of the entire mesh of the problem. The stability of the multigrid method is one of its main hallmarks. This of course has an impact on the computational time in that if the method scaled perfectly, then the time to solve the problem would be constant. Looking at the total column for the multigrid method, we see that the time increases and thus our method is not scaling perfectly. From the table we also see that the relaxation time goes from 70% of the diffusion solution time to 55% at 126 processors while the interpolation time goes from 15% to 23%, and the prolongation time goes from 7% to 11%. What is useful to note is that the interpolation time increases a factor of 6 in our processor range as does the prolongation while the relaxation increases by a factor of 3 with the total diffusion time increasing by a factor of 4. Thus the least scalable parts of our algorithm are the interpolation and prolongation parts.

In the preconditioned conjugate gradient results we note that the number of work units to achieve a solution increases as the problem mesh is refined. In going from 1 to 126 processors, the solution increases by a factor of 4; however, the total computation time increases by a little over and order of magnitude. This is due, in part, to the communications costs with increasing numbers of processors for the inner products involved in the conjugate gradient solution.

3.2 Diffusion Solution of a Small Light Water Problem.

In order to further demonstrate the performance of the parallel multigrid, we have chosen compute a version of a light water reactor as a diffusion problem. This is a 2 group representation of a heterogeneous light water reactor whose spatial layout is depicted in Fig. 7.

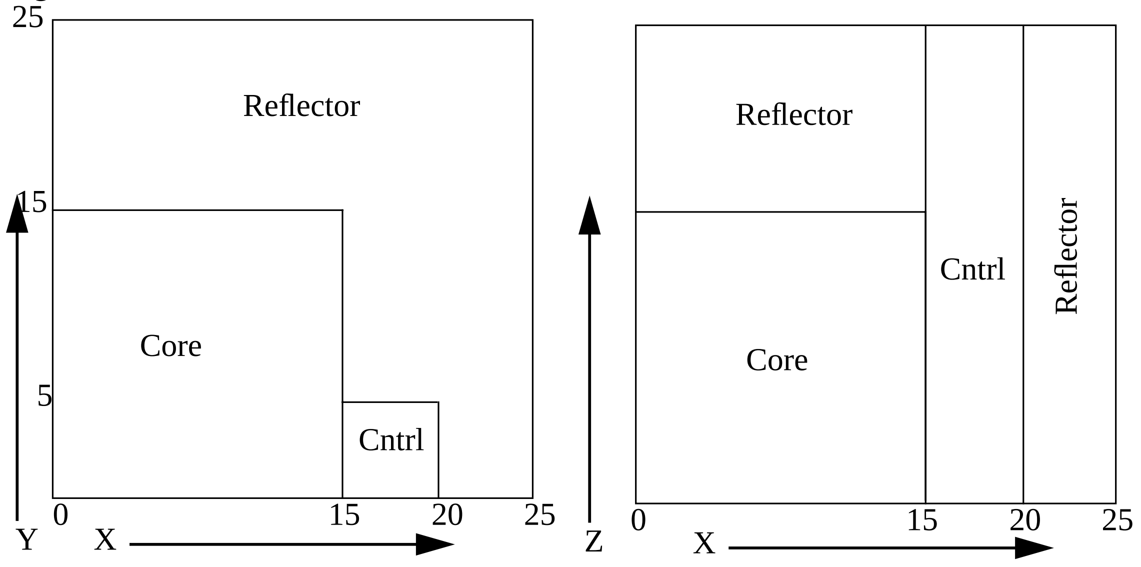

Fig. 7 Schematic of LWR problem for multigrid testing.

In doing a scaling test on this problem, we start from the 1 processor case to mesh it. In this case we use a 1 cm mesh spacing throughout the system resulting in a 25x25x25 mesh. For the multiprocessor cases, we keep the mesh spacing the same while increasing the number of mesh intervals so that the number of mesh intervals per processor is approximately the same. For example, in the 16 processor case we have a 63x63x63 mesh problem which is 15,628 mesh intervals per processor compared to 15,625 for the 1 processor case. We note that the span of the number of mesh intervals in going from 1 to 1008 processors from 1.56e4 to 1.60e7. The LWR scaling problem is run as a Keff computation where the error is iterated down to 1.0e-6 for both Keff and the pointwise fission distribution. We first present the overall results of the scaling calculations and we examine some of the details subsequently. In Table 3, we give the grind time and total run time results for each of the methods as a function of processor count on our ASCI bluemountain platform which is based upon the SGI ORIGIN/2000 architecture.

TABLE 3. Multigrid Performance Data From LWR Problem on the SGI.

	Diffusion Grind Time (ns)				**Diffusion Solution Time (secs)**			
Procs	**CG1L**	**MG**	**MG1l**	**CGMG**	**CG1L**	**MG**	**MG1L**	**CGMG**
1	900.2	895.3	909.4	1246.9	3.9	4.1	3.0	3.0
2	570.1	550.0	552.4	760.6	5.0	5.3	3.7	4.1
4	244.9	334.2	351.4	487.0	4.7	6.2	4.4	5.3
8	135.8	225.7	194.5	269.4	5.8	8.4	5.2	7.4
16	86.8	156.9	113.2	159.5	9.1	12.0	7.0	9.8
32	43.2	88.9	59.8	83.8	11.0	16.6	9.3	12.7
64	21.7	59.4	37.0	52.5	68.8	26.8	12.0	18.2
126	18.6	53.2	26.7	33.1	30.5	54.9	18.0	29.0
252	14.4	39.7	23.9	34.9	53.5	97.6	40.4	186.0
504	7.63	35.8	19.9	27.6	260.5	202.2	106.7	215.4
1008	8.02	37.9	15.6	18.4	672.7	490.6	157.1	340.8

The grind time is the calculational time per spatial mesh cell which in a scaling study should decrease linearly with the number of processors. In examining the grind time results we see that this time does decrease for all methods but with varying rates and levels out for large number of processors. In the second section of Table 3 where we have presented the total calculational time, we see a clear indication of the superiority of the multigrid method vs. the conjugate gradient method. This is because the multigrid method is much more stable in terms of convergence rates than are the CG methods which overcomes the poorer performance in terms of grind times. Also from the MG results we see that doing the 3 line relaxation is not worth the expense in terms of the method's performance. We depict these results graphically in Figs. 8a and 8b.

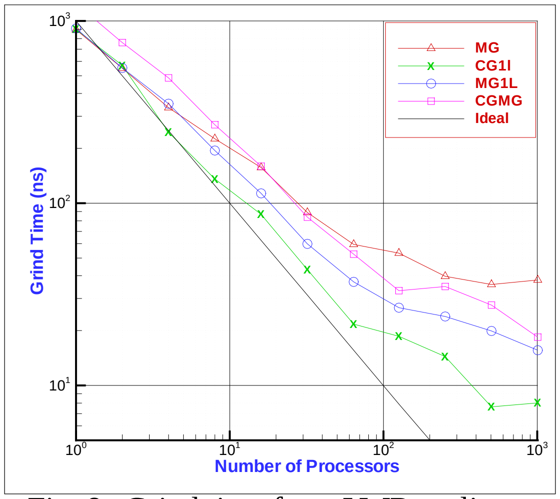

Fig. 8a Grind time from LWR scaling.

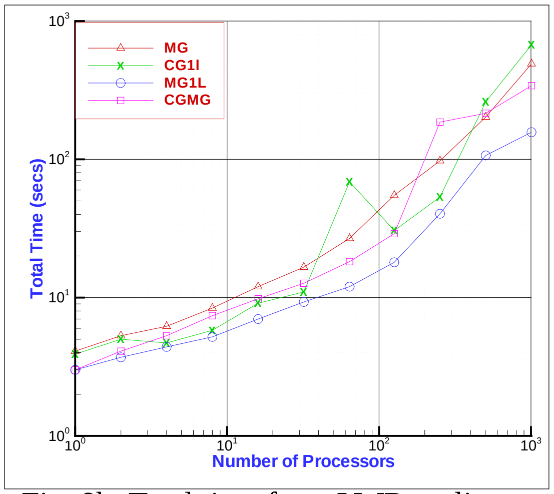

Fig. 8b Total time from LWR scaling.

The results depicted here for the grind time clearly shows that our scaling results are far from ideal especially when we go above 126 processors. This is due to the performance of the MPI communication efficiency on this particular SMP architecture where ‘off-box’ communications have a much higher latency that ‘on-box’ communications which in this case is 128 processors. There is hope that with an improved MPI this situation can be ameliorated. From Fig. 8b the increase in the total computation time with number of processors is partly due to the increase in the grind time but also to the increase in the number of outer iterations needed to converge the problem with more mesh points.

In looking at the components as a percentage of the diffusion solution time in the MG1L method, we note that relaxation goes from 44% to 30% of the total as we go from 1 to 1008 processors. The interpolation time goes from 11% to 15%, the prolongation time goes from 13% to 7%, and the setup time goes from 39% to 40%. Thus as compared to the source driven problem, the fission iteration problem sees a lot more time spent in the setup for the line relaxation than in the relaxation itself!

3.3 A Study of Our Parallelization of the Diffusion Multigrid Method.

We have presented above a results oriented assessment of our particular approach at the parallelization of the inversion of the diffusion operator. In this section we look at what might be loosely termed a computer science evaluation of the same thing. To this end we utilize a tool called Vampir^{TM} from PALLAS GmbH. In this case we do a statistics display which shows the percentage of time on each processor that is spent in the application program (i.e. doing the actual computational work) and the time spent in the various aspects of the MPI communications. In Fig. 9 we give the results from the 64 processor run of the LWR problem using the MG1L method. This is a bar chart where the red bar is for the application, the blue is for MPI barriers, the yellow is for MPI send-receives, the green is for MPI allreduce, and cyan is for MPI receives.

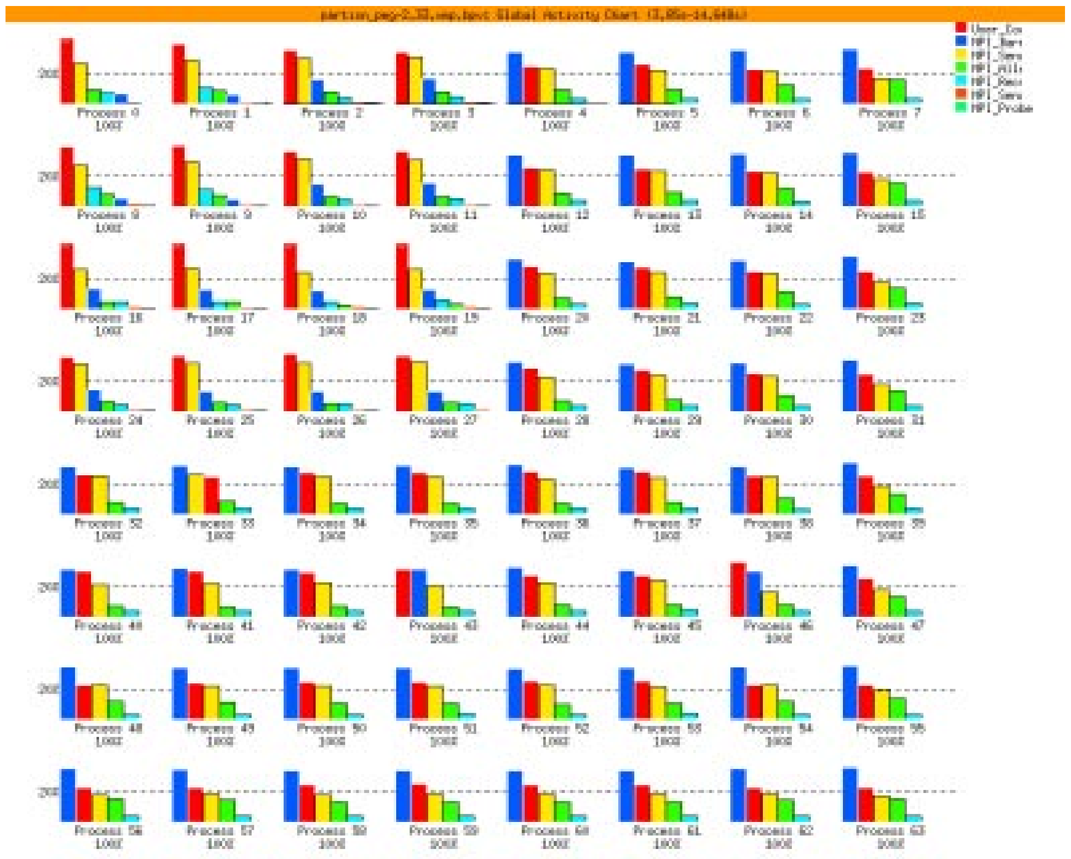

Fig. 9 Vampir Statistics from a 64 Processor MG1L Calculation of LWR.

Recalling that the fine mesh is the only grid on all 64 processors while the next finest grid is on the 16 processors in the left hand corner of the chart, we see the impact of our processor portioning of the grids. The fine mesh work outside of the 16 processors is dominated by the MPI barrier meaning that these processors spend most of their time waiting for work. For all processors the bulk of the work is in the MPI send-receive with the MPI allreduce coming in next at about a third of the time spent in send-receives. The fact that the next finest grid, n-1, communicates with only the 4 processors of grid n-2 is barely discernible in this display of the MPI barrier, meaning that those processors are relatively busy compared with the bulk of the 64. We find this information useful in the future tuning of this method of parallelization suggesting where we should concentrate our efforts of improvement.

4. CONCLUSIONS

We have shown that it is possible to implement a method that makes the multigrid solution of the diffusion equation parallel. We show that it competes well as far as total computation time on representative diffusion problems in reactor physics with the best method we are aware of - preconditioned conjugate gradient. However, it is clear that more needs to be done to make the multigrid method efficient on hundreds to thousands of processors on the current SMP architectures. Communication is a significant cost for multigrid and so other means of effecting communication between grids needs to be investigated. But given the remarkable stability of the multigrid method itself on typical neutronics problems, we deem the effort to make it efficient on thousands of processors a worthy goal. To this end we are working at reducing the setup costs since they figure prominently in neutronics problems with fission and/or upscatter or those that are time dependent. We are also seeking ways to speed up the computations on the coarser grids so as to avoid making the fine mesh processors wait so long to compute.

REFERENCES

Alcouffe, R. E., Brandt, Achi, Dendy, J. E. Jr, and Painter, J. W., 1981. The Multigrid Method for the Diffusion Equation with Strongly Discontinuous Coefficients, SIAM J. SCI. STAT. COMPUT., Vol 2, No. 4, 430-454.

Alcouffe, R. E., 1983. The Multigrid Method for Solving the Two-Dimensional Multigroup Diffusion Equation. In: Advances in Reactor Computations - Proceedings of a Topical Meeting. Salt Lake City, pp340-351.

Alcouffe, R. E., Baker, R. S., Dahl, J. A., and Turner, S. A., 2000. PARTISN Code Abstract. In: Physor 2000 International Topical Meeting, Advances in Reactor Physics and Mathematics and Computation into the Next Millennium, Pittsburgh.

Baker, R. S., and Alcouffe, R. E., 1997. Parallel 3-D Sn Performance for DANTSYS/MPI on the Cray T3D. In: Joint International Conference on Mathematical Methods and Supercomputing for Nuclear Applications, Saratoga NY.